2005－QAI－14（16）
2005／1／19

レートに基づく複数経路通信プロトコルR-M/TCPにおけるデータ分配手法

大須賀　徹†　　ロットウィブンチャイグンティダー††　　相田　仁††

† 東京大学大学院 工学系研究科
〒 113–0033 東京都文京区本郷 7–3–1
†† 東京大学大学院 新領域創成科学研究科
〒 277–8561 千葉県柏市柏の葉 5–1–5
E-mail: †{t-osuga,kultida,aida}@aida.k.u-tokyo.ac.jp

あらまし　従来のTCPにおけるロングファットパイプ問題やLoss Recovery Algorithm，バーストトラヒックや経路障害に対する脆弱性などの問題を解決するため，我々は各パスの利用可能帯域を見積もり，それに応じて送信レートを調節する複数経路通信プロトコルR-M/TCPを提案している．これにおいて，コネクション内の各パスの状態が大きく異なる場合，受信ウィンドウ内に確認応答の取れないデータが残り続けてしまうことで，全体の通信効率が性能の低いパスの影響を強く受けてしまう問題を改善するため，各パスのスループットや遅延に応じてパケットを分配することでパケットの受信側への到着順序違いを抑える手法を提案し，シミュレーションを行ってその性能を評価し，本提案手法の効果を確認した．

キーワード　データ分配，送信レート制御，複数経路，TCP

Packet scheduling algorithm for Rate-based Multi-path TCP (R-M/TCP)

Toru OSUGA†, Kultida ROJVIBOONCHAI††, and Hitoshi AIDA††

† School of Engineering, The University of Tokyo
Hongo 7–3–1, Bunkyo-ku, Tokyo, 113–0033 Japan
†† Graduate School of Frontier Sciences, The University of Tokyo
kashiwanoha 5–1–5, kashiwa-shi, Chiba, 277–8561 Japan
E-mail: †{t-osuga,kultida,aida}@aida.k.u-tokyo.ac.jp

Abstract　We have proposed Rate-based Multi-path Transmission Control Protocol (R-M/TCP) as a reliable transport protocol to tackle with the well-known long-fat-pipe problem, loss recovery algorithm, and burst traffic in TCP. R-M/TCP performs congestion control in a rate-based and loss-avoidance manner. It originally transmits data packets in a round-robin manner through multiple paths. In this paper, instead of using the round-robin manner, we propose a packet scheduling algorithm for R-M/TCP. Our algorithm accounts for the time-varying bandwidth and delay of each path so as to minimize number of packets arriving in out-of-order at the receiver. Simulation results show effectiveness of our algorithm.

Key words　Packet scheduling, Rate Control, Multipath, TCP

1. はじめに

現在，ファイル転送などの多くの通信で用いられているTCP[1],[2]は，送信量を常に増やしていき，損失が起きて初めてcwnd（Congestion Window，輻輳ウィンドウ）を半分にするという極端な対処（Loss Recovery Algorithm）をしているため，送信量（cwnd）の起伏が激しく不安定で無駄も多くなってしまうという問題がある．

また，コネクションがロングファットパイプのとき，利用可能な帯域を使いきれるまで時間がかかってしまうので，この問題と前述のLoss Recovery Algorithmにより，実際のネットワーク状態への追従が遅くなってしまうという問題がある．

また，従来のTCPでは，ACK（ACKnowledgement，確認応答）が返って来たときにフロー制御，輻輳制御が許す限り全てバースト的にパケットを送り，それ以外のときは送らないでいるため，緩慢の激しいバーストトラヒックによって不必要な輻

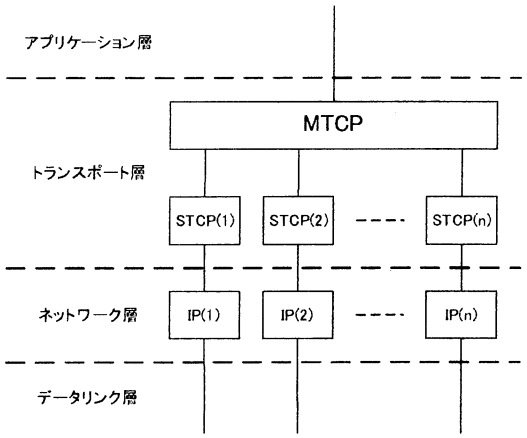

図1 M/TCP のレイヤ構成

輻輳や損失が起きやすくなってしまっている，といった問題点を抱えている．

これらの問題に対し，我々は，ACK の到着状況などによりそのパスで利用可能な BW（Band Width，帯域）を見積もり，また各パス上に溜まっていると予測されるキューの量を推定して，損失が起きる前に見積もった BW にあわせて cwnd を制限することで損失を回避する Congestion Control，また，RTT（Round Trip Time，周回遅延）内で送信レートが平滑的になるように一定間隔でパケットを送信する Rate Control などを用いてこれらの問題を解決する複数経路通信プロトコル R-M/TCP を提案している [5]．

しかし，このプロトコルはコネクション内の各パスへパケットを分配する際に，各パスの状態に関わらず均等に分配しているため，性能差の激しいパスが混在する環境においては複数パスの通信効率を出し切れていないという問題がある．

そこで本稿では，コネクション内の各パスの状態の差異によるスループットの低下を改善するため，各パスの状態に応じてパケットを分配することで受信側へのパケットの到着順序違いを抑える手法を提案し，ネットワークシミュレータの ns-2 [6] においてその性能を評価した．

2. レートに基づく複数経路通信プロトコル R-M/TCP

2.1 複数経路通信プロトコル M/TCP [3]

M/TCP は，エンド−エンド間における 1 つの TCP コネクションの中に複数のパスを張ることで複数経路通信を実現するプロトコルで，図 1 のようなレイヤ構成をしており，トランスポート層を 2 つの副層に分けて輻輳制御は下位副層にある各々のパスが行い，フロー制御や順序制御は上位副層が一括して行っている．

また，1 つのコネクション内に複数のパスを張るため，図 2 のような 8 [Byte] のマルチルートオプションを TCP オプションフィールドに設けて送受信に用いたパスの識別を行っている．

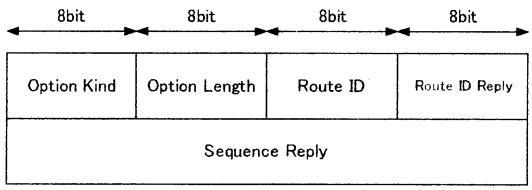

図2 M/TCP マルチルートオプション

また，パスが複数あるため往路と復路で違うパスを通ることがあるので，パケットの送受信にかかった時間には RTT ではなく，片道遅延をあらわす OWTT（One way trip time）を用いている．

2.2 R-M/TCP の概要と機構

前節で述べたように，TCP にはロングファットパイプ問題や Loss Recovery Algorithm，バーストトラヒックといった問題があり，TCP をベースに複数経路通信を行っている M/TCP も同様の問題を抱えている．

これらの問題に対し，我々は各パスの利用可能帯域を見積もり，それに応じて送信レートを調節する複数経路通信プロトコル R-M/TCP を提案している．

以下の 2.2.1 項と 2.2.2 項で，TCP の問題を解決するための機構を紹介していく．

2.2.1 Congestion Control

本提案手法の Congestion Control ではロングファットパイプ問題や Loss Recovery Algorithm により，cwnd の変動が激しく，また実際のネットワーク状況に対して追従性が遅いという問題の解決をめざしている．

まず，以下のような式によって各パスで利用可能な帯域 $bw\_[id]$ を見積もる（BW Estimation）．ここで id は複数あるパスの ID をあらわす．また，$OWTTf\_[id]$ は往路の OWTT をあらわし，$OWTTr\_ave$ は復路の全パスの OWTT の平均値をあらわす．そして，num_acked は $RTT\_[id]$ の間に ACK されたパケットの個数をあらわすものとする．

$$RTT\_[id] = OWTTf\_[id] + OWTTr\_ave \quad (1)$$

$$bw\_[id] = \frac{num\_acked}{RTT\_[id]} \quad (2)$$

また，以下のような式で各パス上に溜まっていると予測されるキューの量 $qlen\_[id]$ を推定する（Queue Length Estimation）．ここで $RTT\_min\_[id]$ は通信開始から現在までの $RTT\_[id]$ の最小値をあらわすものとする．

$$qlen\_[id] = bw\_[id] \times (RTT\_[id] - RTT\_min\_[id]) \quad (3)$$

式 (3) で求めた $qlen\_[id]$ が増えると，そのパス中にキューが溜まってきておりいずれパケット損失が起きると予想されるので，以下のような式で $cwnd\_[id]$ と Slow-Start Threshold 値である $ssthresh\_[id]$ を変更する（Congestion Control）．

$$cwnd\_[id] = bw\_[id] \times RTT\_min\_[id] \quad (4)$$

$$ssthresh\_[id] = bw\_[id] \times RTT\_min\_[id] \quad (5)$$

このような流れでパケット損失が起きる前に cwnd を素早くネットワークの状態に追従させることで，この項の冒頭で述べたような問題を解決することができる．

2.2.2 Rate Control

ACK が返って来たときにフロー制御，輻輳制御が許す限りバースト的にパケットを送ってしまっているため，不必要な輻輳や損失が起きやすくなってしまっているバーストトラヒックを解決する機構が Rate Control である．

まず，1 回の RTT 中の送信回数 $burst\_limit\_[id]$ と，送信間隔 $burst\_interval\_[id]$ を以下のような式で求める．ここで，1 回の送信で同時に送るパケットの個数を $burst$ とする．現在，ボトルネックリンクの帯域をパケットペアで測定するため，$burst$ は 2 個に設定している．

$$burst\_limit\_[id] = \frac{cwnd\_[id]}{burst} \quad (6)$$

$$burst\_interval\_[id] = \frac{RTT\_[id]}{burst\_limit\_[id]} \quad (7)$$

これによって導き出された $burst\_interval\_[id]$ の間隔で $burst$ 個ずつパケットを送信することで，cwnd 分のパケットを 1 回の RTT 中に均等に分散させて送出することができるようになる．

また，ネットワークの変化に追従していくため，cwnd が大きく変わるたびに $burst\_interval\_[id]$ を計算し直している．

3. R-M/TCP におけるデータ分配手法

3.1 従来の R-M/TCP の問題点

従来の R-M/TCP は，パケットの送信方法が cwnd や受信ウィンドウの許す範囲でコネクション内の各パスに順々にパケットを振り分ける巡回方式であるため，各パスの遅延などに関係なくそれぞれのパスに均等にパケットが振り分けられてしまう．

そのため，各パスの状態（遅延や帯域など）が大きく異なるネットワーク環境において，スループットの高いパスからのパケットが次々と先に届いていても，櫛状に所々届いていないスループットの低いパスからのパケットを待たなければ ACK を更新することができないため，性能の低いパスに足を引っ張られて複数パスのスループットを活かしきれていないという問題がある [5]．

この問題に対し，本稿では各パスの状態に応じてパケットを分配し，受信側に順序番号通りに届くようにすることで，まだ確認応答されずに受信ウィンドウ内に確保されているデータ容量を減らして受信ウィンドウサイズを最大限有効活用し，各パスの状態が大きく異なる環境におけるスループットの低下を抑える手法を提案する．

複数経路通信プロトコル M/TCP においてデータ分配を行う研究 [4] と同様に，送信側で得られる情報を基にパケットの受信側への到着時間を予測して，順序番号通りに届かせることで順序番号の高い ACK を返させることを主目的としている．

このアルゴリズムの詳細を 3.2 項で紹介する．

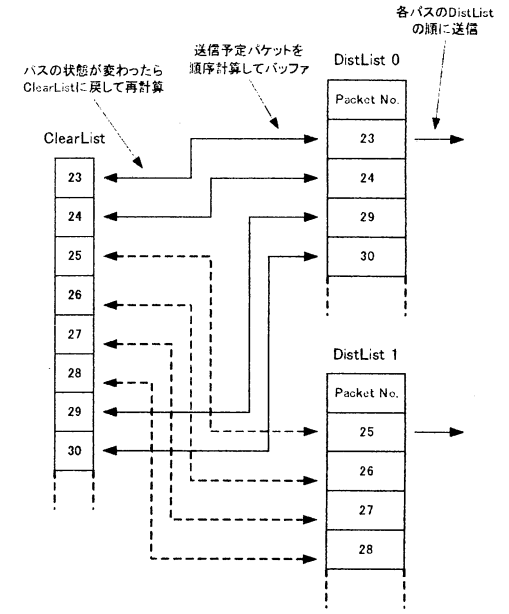

図 3　分配リストとクリアリスト

3.2 データ分配アルゴリズム

M/TCP においてデータ分配を行う研究と大きく異なる点は，R-M/TCP では Rate Control によりパケットの送信を一括で行わないこと，そして各パスで別々のタイミングでパケットを送信することである．そのため，M/TCP におけるデータ分配のように，同時に一括で送信されるパケットの範囲内で送出する順序だけを入れ替えるということはできない．

そこで，図 3（便宜上，順序番号でなくパケットの番号で記述している）のような 2 種類のリストを新たに送信側に設けた．それぞれ DistList（Distribution List，分配リスト）と ClearList（クリアリスト）という名称で，DistList は Rate Control によって一括に送信されずに $burst\_interval\_[id]$ の間だけ待つことになる送信予定分のパケットの順序番号をバッファしておくためのリストである．また，ClearList は各パスの状態などが大きく変わってパケット分配をし直す必要性が出てきたときに，各パスの DistList の中身を統合して一時的に退避させるためのリストである．

ここで，パケットの分配，再分配の流れは以下のようになっている．

（1） パケット分配の判別式（後述の式 (9)，(10)）に従って各パスの DistList にパケットを分配し，$burst\_interval\_[id]$ ごとに DistList の先頭にある順序番号のパケットを送信する．

（2） 各パスの状態が変わって Congestion Control を再び行ったり，送信間隔である $burst\_interval\_[id]$ が大きく変わったり

したときには，到着順序が変わってくるのでパケットの分配をやり直す．

（3） パケット分配の判別式を再計算するにあたって，各パスの DistList の中身を統合して ClearList に退避させる．このとき分配し直す量が少なくならないように，まだ ClearList に入ってなかったパケットの順序番号を ClearList へ補充する．

（4） パケット分配の判別式を計算し直した後，それを基に ClearList から再び DistList にパケットを分配する．

次にパケット分配の判別式について紹介する．M/TCP におけるデータ分配では，各パスの $OWTTf\_[id]$ と受信側への ACK の到着間隔 $inter\_[id]$ を用いて

$$OWTTf\_[id] + inter\_[id] \times i \tag{8}$$

が i 番目のパケットの到着予定時間になり，この値の小さい方のパスが先に到着すると予測されるので，これを基にパケット分配を行っている [4].

R-M/TCP のデータ送信は 2.2.2 項にあるように，Rate Control によって $burst\_interval\_[id]$ ごとにパケットを 2 個ずつパケットペアで送っており，また，それぞれのパスで別々の $burst\_interval\_[id]$ を基にパケット送信を行っているため，同時にパケットが送信されているわけではない．

そこで，現在時刻を now，前回のパケット送信時刻を $last\_sent\_[id]$，パケットペアの到着間隔を $inter\_[id]$ とすると，

$$\text{奇数番目}：now - last\_sent\_[id] + burst\_interval\_[id] \times i + OWTTf\_[id] \tag{9}$$

$$\text{偶数番目}：now - last\_sent\_[id] + burst\_interval\_[id] \times i + OWTTf\_[id] + inter\_[id] \tag{10}$$

が $2 \times i - 1$ 番目と $2 \times i$ 番目のパケットの到着予定時間になり，これを基にパケット分配を行う．

4. シミュレーションによる評価

4.1 シミュレーション環境と結果

ネットワークシミュレータの ns-2 を用いて，従来の R-M/TCP と提案手法の比較評価を行った．シミュレーションは図 4 のような環境で行った．このとき，受信ウィンドウサイズは 64 [KB] としている．

図 4 のネットワークのモデルにおいて下のパスの OWTT を 30 [ms] ～ 120 [ms] まで 15 [ms] 刻みで変化させ，その各々の環境で 100 [s] の間データ送信を行い，5 つの R-M/TCP コネクションが送ったデータ量の平均を取ってスループットを求めた．この際，シミュレーション途中の 30 [s] ～ 60 [s] の間，Node8 から Node9 へと Node10 から Node11 へ TCP-Reno による通信を 5 コネクションずつ入れてクロストラヒックとした．その結果を図 5 に示す．

また，図 4 の環境はロングファットパイプとなるので，ロングファットパイプの環境下でのスループットである

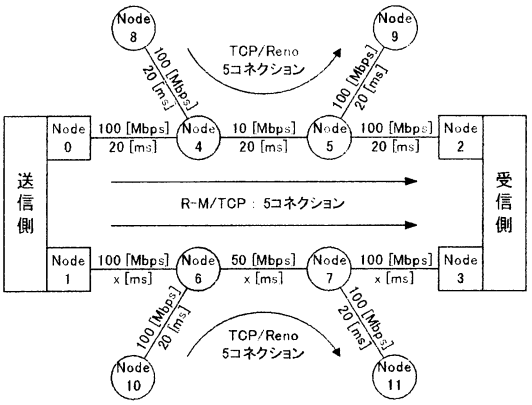

図 4 シミュレーション環境
(x は 10 [ms] ～ 40 [ms] を 5 [ms] 刻みで増やす)

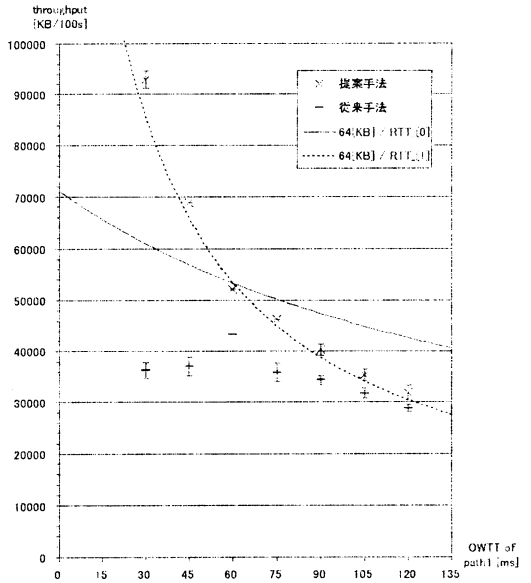

図 5 100 秒間におけるデータ通信量

$$\frac{\text{受信ウィンドウサイズ}}{RTT\_[id]} \tag{11}$$

の曲線を図 5 に加えた．

また，パケットの受信側への到着順序が大きく変わると予想される．下のパスの OWTT が 30 [ms] の環境において，測定の際に受信側に届いたデータの順序番号をモニタすることで，まだ確認応答されていないために受信ウィンドウ内に確保されているデータ容量を集計し，従来手法と本提案手法を比較した．その結果を図 6，図 7 に示す．

4.2 考　　察

R-M/TCP は 2 節で紹介した Congestion Control や Rate Control などの諸々の機能により，各パスの状態に応じた輻輳制御

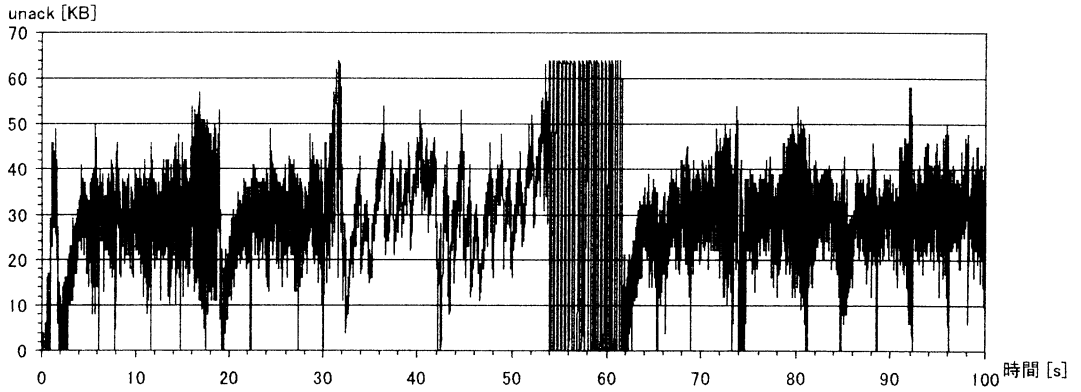

図6　受信ウィンドウ内に残る未 ACK 分のデータ領域（従来手法）
（測定環境：図 4 において $OWTTf\_[1] = 30$ [ms] としたとき）

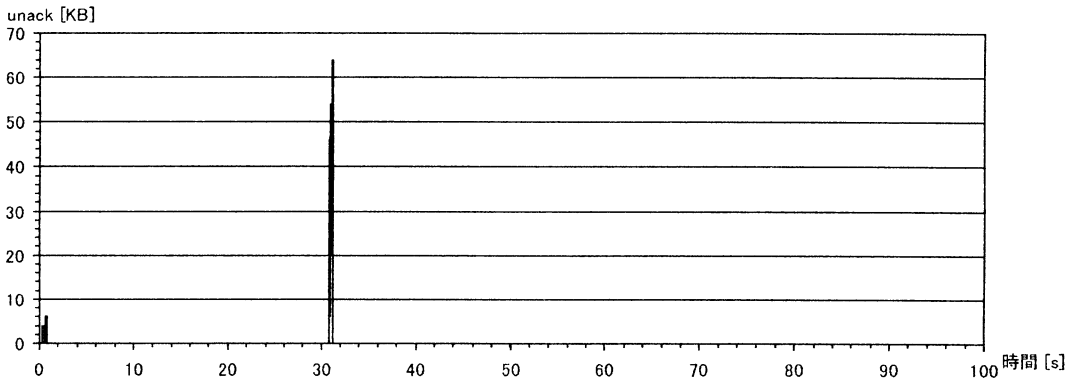

図7　受信ウィンドウ内に残る未 ACK 分のデータ領域（提案手法）
（測定環境：図 4 において $OWTTf\_[1] = 30$ [ms] としたとき）

を早い追従性をもってかけているため，性能の低いパスの影響を M/TCP に比べて低く抑えることができ，また受信ウィンドウ内に十分な余裕がある場合は ACK の取れていないデータがある程度残っていても，全体のスループットにはあまり影響を及ぼさずに済むことができる．

しかし，各パスの遅延が大きかったり，また帯域が十分に広かったりなど，スループットの上限が cwnd よりも受信ウィンドウの空き容量に左右されるような状況（ロングファットパイプ）になると，R-M/TCP も各パスの状態の差異の影響を大きく受けてしまい，スループットが上がらなくなってしまう．

その環境にて行ったシミュレーションにおいて，図 5 より提案手法は各パスの状態の違いの影響を受けず，スループットを式 (11) の理論上限値まで引き出しているのが分かる．それに対し，従来手法はたとえ片方のパスの OWTT が小さくなっていってもパケットが受信側に櫛状に届いてしまい，OWTT の大きいハスからのパケットを待つことになるので図 5 のようにスループットが頭打ちになってしまうことが分かる．

このことより，例えばコネクションの中の一部のパスがロングファットパイプであるような環境において，従来手法ではそれらのパスの影響をコネクション全体が受けてしまい，一番遅いパスの遅延を式 (11) に適用したスループットで頭打ちになってしまう．それに対し本提案手法では，各パスの性能に合わせて送信するパケットの順序番号を割り振っているので，性能の高いパスも低いパスも含めた全体の性能和を使うことができると考えられる．

ここで，図 5 において提案手法のスループットが式 (11) の理論曲線を上回っている部分があるが，図 5 に引いた理論曲線は式 (1)，(11) より

$$\frac{\text{受信ウィンドウサイズ}}{OWTTf\_[id] + OWTTr\_ave} \tag{12}$$

の曲線なので，これは ACK が平均的に $OWTTr\_ave$ で返ってきたときの理論スループット上限となる．しかし実際は，速い方のパスで高い ACK 値が返ってきた場合，送信側は遅い方のパスから返ってくるそれ以前の ACK 値を待つことなく受信ウィンドウを更新していくので，理論値は $OWTTr\_ave$ よりも速い方の OWTT の値に近づくことになり，その結果スループット

が式 (11) の曲線を上回っていると考えられる.

また，下のパスの OWTT が大きいときに提案手法と従来手法の差がそれほど出ていないのは，OWTT が大きく送信に時間がかかると，途中で $burst\_interval\_[id]$ の値が変わってしまうことが多く，そのため提案手法でも受信側に順序違いで届くパケットの量が増えてしまい，正確に順序通り届けることができなかったことが原因と考えられる.

また，各パスの OWTT が等しくなる 60 [ms] のところでは，パスの状態の差異がほとんど無くなってしまう（ボトルネックリンクの帯域の差だけ）ので，データ分配の効果が薄くなり，図 5 のように各手法の差が小さくなっていると考えられる.

次に，図 6，図 7 より，従来手法の図 6 ではまだ確認応答がされず受信ウィンドウ内にデータ容量が常に多く確保されて受信ウィンドウサイズを圧迫していたのに対し，提案手法の図 7 では，受信ウィンドウ内に確保されて残っているデータ容量はほとんど無く，データが順序番号通りに受信側に届いていることが分かる．図 6，図 7 は便宜上極端な状況でのシミュレーション結果を例として載せたが，他の環境においても同様に順序違いを抑えることができており，これにより，図 5 と併せて本提案手法の効果が出ていることが確認できた.

また，図 7 において，一時的に受信ウィンドウ内に確認応答されていないデータが突出してしまっているのは，そこがネットワークの状態の変わり目である送信開始直後とクロストラヒックが出現した直後であり，データ分配の判別式が追従しきれずに順序違いが起きてしまったためだと考えられる.

このシミュレーションを行った現在，本提案手法の問題点として考えられるのは，前述のように R-M/TCP の Congestion Control や Rate Control などの機能により，いずれかのパスにおいて受信ウィンドウの空き容量がネックとなるような環境でないと本提案手法の効果があまり出てこないことが挙げられる．それに加えて，Congestion Control がパケット損失を起こさないために cwnd にネガティブな（強く抑え気味の）制限をかけているため，受信ウィンドウサイズよりも小さく抑えこまれた cwnd がネックとなってしまう通信になりやすいという問題がある.

特に後者の問題に対しては，各パスの中に送信途中のパケットがどの程度溜まっているかを監視し，その値が小さくパスに余裕があると予測されるときは，そのパスのボトルネックリンクの帯域を考慮して，BW Estimation によって見積もられている帯域の値を増やしてアクティブに cwnd を向上させていく，というように Congestion Control を改善していく必要があると考えられる.

5. ま と め

本稿では，R-M/TCP において，各パスの状態に応じてパケットを分配し，受信側に順序番号通りに届くようにすることで，確認応答されずに受信ウィンドウ内に確保されているデータ容量を減らして受信ウィンドウサイズを最大限有効活用し，各パスの状態が大きく異なる環境におけるスループットの低下を抑えるという手法を提案した.

また，ネットワークシミュレータの ns-2 で本提案手法と従来の R-M/TCP との比較評価を行い，本提案手法が受信側へ順序違いで届いてしまうパケットの量を抑え，各パスの状態が大きく異なる環境においてもスループットの低下を防ぐことができるということを確認した.

今後の課題としては，4.2 項で述べたように，Congestion Control を改善して各パスの中に送信途中のパケットがどの程度溜まっているかを監視し，パスに余裕があると予測されるときは，そのパスのボトルネックリンクの帯域を考慮して，帯域の見積もり値を増やして cwnd を向上させていくことが挙げられる.

文　献

[1] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," IETF RFC 2581, Apr. 1999.

[2] V. Paxson, and M. Allman, "Computing TCP's retransmission timer," IETF RFC 2988, Nov. 2000.

[3] K. Rojviboonchai, and H. Aida, "An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes," IEICE Trans. on Commun., vol.E87-B, no.9, pp.2699-2707, Sept. 2004.

[4] K. Rojviboonchai, T. Osuga, and H. Aida, "Delay time-based data transmission sequence for multi-path transmission control protocol (M/TCP)," Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM '03, Aug. 2003.

[5] K. Rojviboonchai, T. Osuga, and H. Aida, "R-M/TCP : protocol for reliable multi-path transport over the Internet," The IEEE 19th International Conference on Advanced Information Networking and Applications, AINA 2005, Mar. 2005 (accepted).

[6] "The Network Simulator -ns2-", http://www.isi.edu/nsnam/ns/